Rutgers University Electrical and Computer Engineering Capstone
Neuro-Kinetic Virtual Control System

Advisor: Professor Dario Pompili
Group 4: Asim Alvi, Frank Bohn, Giovanni DeGrande, Eric Lee, Sangit Patel

Abstract:

Many people with different forms of paralysis and degenerative diseases, such as Muscular Dystrophy, Osteoarthritis, and Diabetes struggle with interacting with computers and control systems. Simple things such as turning on and off lights, moving, and using computer applications are extremely difficult. Our goal was to combine the use of electroencephalograms and kinematic technology to control some of the aforementioned systems. To prove this concept we used a shimmer sensor. The shimmer sensor has a built in accelerometer and gyroscope. The sensor is attached to the user's head and through various movements they can activate particular keys on a keyboard. Through the use of an EEG data we would be able to obtain a larger number of control signals than interpreting accelerometer and gyroscope data. Our proof of concept uses a shimmer sensor to simulate keystrokes on the computer. We use these keystrokes to play a game of Mario. Current technology focuses on one input method. They either use EEG data or gyroscopic and accelerometer data. Our method improves on this by harnessing both methods and requires only a working head and neck to implement. There is a lot of room to expand on what we did. Future work could include improving the methods of processing the data. It could also include utilizing more mathematical techniques in order extract more artifacts from the brainwaves. Venturing out in to several other platforms can prove to be very advantageous. The ultimate goal for future work would be to implement a universal system that can be easily implemented into modern electronics that will allow people to communicate with multiple devices at once. This will allow them to have greater control over their environment.

Overview:

1. Goals:

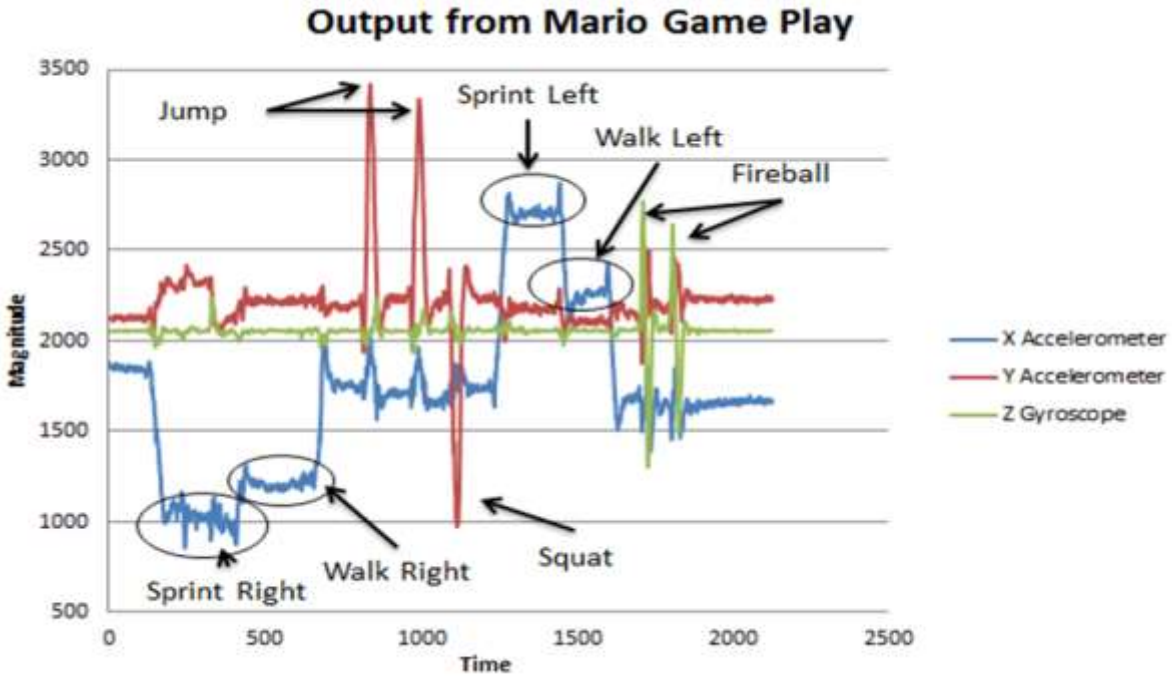
Our main goal was to create a way for people suffering from paralysis or degenerative diseases to be able to control devices using simple head movements. Using this control system, we aimed to control devices such as motorized wheelchairs. As a proof of concept, we scaled down our goal to implement a game using a Shimmer Sensor, which is an accelerometer and gyroscope (See Below). We collected the data from the sensor and then processed the data into controls.

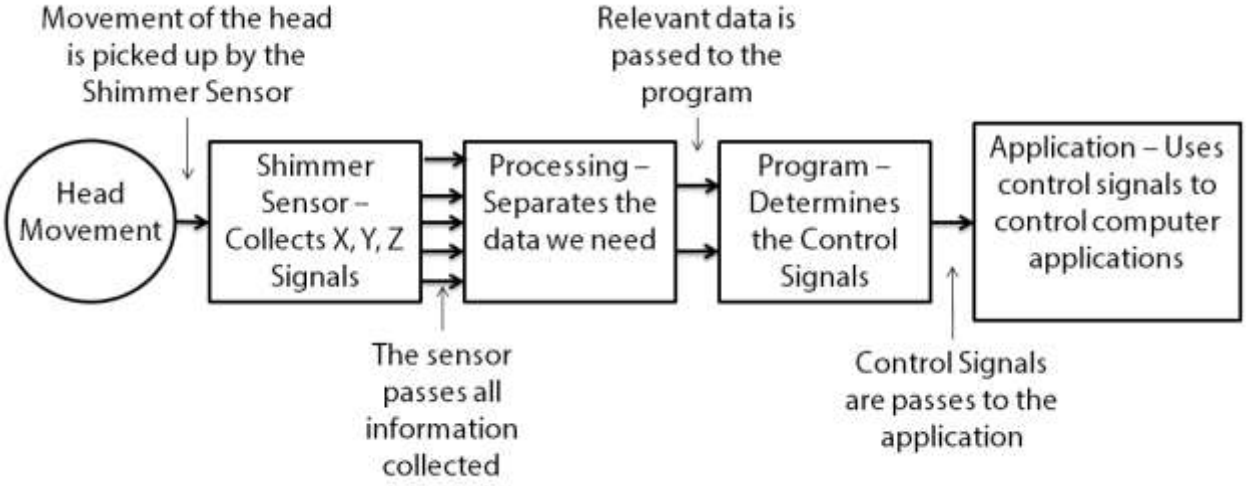
2. Motivations:
The current systems in place for people with limited mobility do not satisfy their needs for a reliable and affordable system that can be used to control everyday things. Our motivation was to fulfill these needs.

3. Objectives:
Our primary objective is to have real-time control of a computer game while processing and immediately using the live data outputted by a Shimmer Sensor. Next, our objective was to broaden the scale and apply that control to many other programs and devices.

4. Research Challenges:
Due to unforeseen circumstances, we were unable to procure a Electroencephalography (EEG) device. In order to work around this, we decided to use an accelerometer and gyroscope.
Challenges arose when figuring out a way for the Shimmer Sensor to communicate with the program in real time, due to the logging rate of the sensor and data processing time. Due to various forms of values that were output from the sensor, we found it very difficult to properly parse the data.

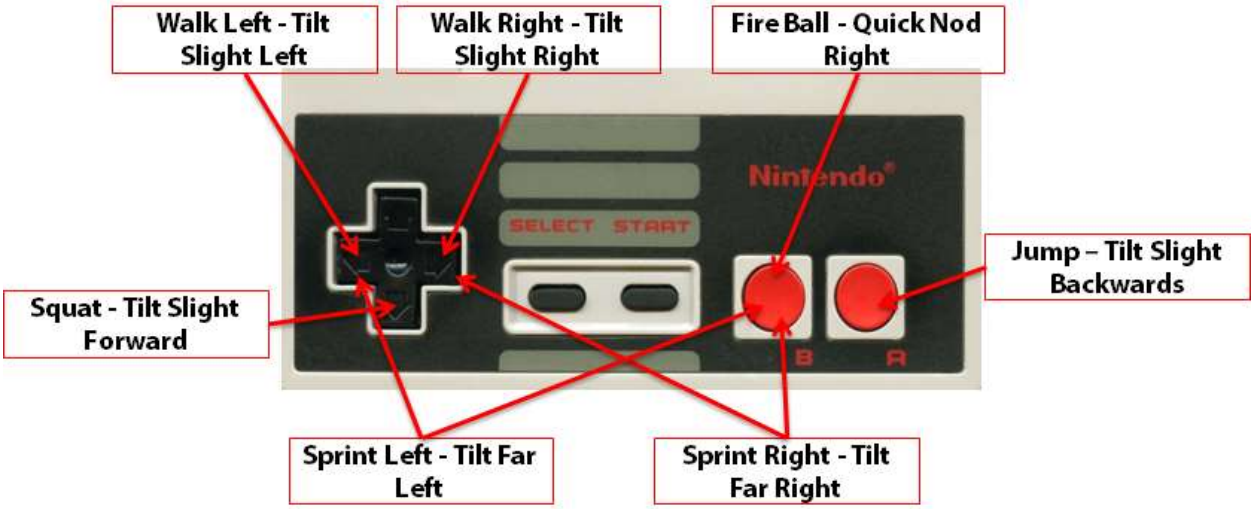
5. Methodology:
Using the Shimmer Sensor, we first collect multiple signals from the x, y, and z axis. The sensor's data is parsed and processed to separate accelerometer and gyroscope data, which is then passed into the program. Based on certain parameters, the values of the accelerometer and gyroscope are used to determine control signals. These control signals are then used to simulate keystrokes, which are then used to control the various computer applications, such as games.



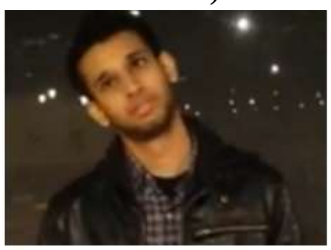

Movement of the head is picked up by the Shimmer Sensor
Relevant data is passed to the program
Head Movement
Shimmer Sensor – Collects X, Y, Z Signals
Processing – Separates the data we need
Program – Determines the Control Signals
Application – Uses control signals to control computer applications
The sensor passes all information collected
Control Signals are passes to the application

6. Results

Computer Game Results – We found that a user with limited mobility would successfully be able to play the computer game with only using the movement of their head. The control signals corresponded to the following movements:


Walk Left - Tilt Slight Left
Walk Right - Tilt Slight Right
Fire Ball - Quick Nod Right
Jump – Tilt Slight Backwards
Squat - Tilt Slight Forward
Sprint Left - Tilt Far Left
Sprint Right - Tilt Far Right
Nintendo
SELECT START
B
A

(This diagram show a Nintendo controller and which head movements correspond to which buttons in Super Mario Brothers. This control system can also be applied to other games with similar controls.)



Tilt Head Right – Run Right

Tilt Head Up – Jump

Tilt Head Left – Run Left

(This diagram shows an example of the head movements needed. A better description of the head movements are below.)

Movement in Game	Corresponding Buttons	Head Movement
Walk Left	Left	Tilt Head to the Left (<20 degrees)
Run Left	Left and B	Tilt Head to the Left (>20 degrees)
Walk Right	Right	Tilt Head to the Right (<20 degrees)
Run Right	Right and B	Tilt Head to the Right (>20 degrees)
Jump	A	Tilt Head Back (>15 degrees)
Squat	Down	Tilt Head Forward (>15 degrees)
Fireball	B	Quick Head Nod to the Right

Our most important result was proving our concept. With just a few control signals and one shimmer sensor, we have shown that is it possible to have someone with very limited mobility control a computer game. This control system can be scaled to many more

applications and can be used to control numerous devices. With further research and development, it is possible to create an intensive control system that will greatly increase the capabilities of those with limited mobility and increase their value of life.

Methods and Results

As explained before, the Shimmer sensor measures accelerometer and gyroscopic data which is then written to a file via Shimmer's proprietary control program. Our first challenge was with reading the data from this file, more specifically, how to read only the data we needed in order to reduce memory overhead.

Shimmer 1 Timestamp	Shimmer 1 Accelerometer X	Shimmer 1 Accelerometer Y	Shimmer 1 Accelerometer Z	Shimmer 1 Gyroscope X	Shimmer 1 Gyroscope Y	Shimmer 1 Gyroscope Z	Shimmer 1 Timestamp	Shimmer 1 Accelerometer X	Shimmer 1 Accelerometer Y	Shimmer 1 Accelerometer Z	Shimmer 1 Gyroscope X	Shimmer 1 Gyroscope Y	Shimmer 1 Gyroscope Z
RAW	RAW	RAW	RAW	RAW	RAW	RAW	CAL	CAL	CAL	CAL	CAL	CAL	CAL
No unit	No unit	No unit	No unit	No unit	No unit	No unit	mSecs	m/(sec^2)*	m/(sec^2)*	m/(sec^2)*	deg/sec*	deg/sec*	deg/sec*
49569	1941	2101	2911	2045	1999	2056	1512.72583	-1.059405941	0.524753475	8.544554455	-57.14285714	-73.99267399	-78.02197802
50209	1951	2090	2905	2046	1997	2053	1532.25708	-0.96039604	0.415841584	8.485148515	-56.41025641	-74.35897436	-76.92307692
50849	1954	2090	2898	2043	1997	2055	1551.78855	-0.950693069	0.415841584	8.415841584	-56.41025641	-72.26007326	-77.65567766
51489	1947	2088	2899	2043	1997	2054	1571.31958	-1	0.396039604	8.425742574	-56.41025641	-73.26007326	-77.28937729
52129	1947	2095	2893	2046	2000	2053	1590.85083	-1	0.465346535	8.366336634	-57.50915751	-74.35897436	-76.92307692
52769	1952	2091	2900	2044	1999	2055	1610.38208	-0.95048505	0.425742574	8.435643564	-57.14285714	-73.62637363	-76.92307692
53409	1946	2102	2910	2049	1999	2054	1629.91333	-1.00990099	0.534653465	8.534653465	-57.14285714	-75.45787546	-77.28937729
54049	1961	2083	2908	2045	1996	2054	1649.44458	-0.861386139	0.346534653	8.514851485	-56.04395604	-73.99267399	-77.28937729
54689	1951	2087	2907	2047	1999	2055	1668.97585	-0.96039604	0.386138614	8.504950495	-57.14285714	-74.72527473	-77.65567766
55329	1941	2104	2894	2045	1997	2053	1688.50708	-1.059405941	0.554455446	8.376237624	-56.41025641	-73.99267399	-76.92307692
55969	1959	2088	2896	2045	1997	2053	1708.03833	-0.881188119	0.396039604	8.396039604	-56.41025641	-73.99267399	-76.92307692
56609	1957	2083	2903	2045	1997	2055	1727.56958	-0.900990099	0.346534653	8.465346535	-56.41025641	-73.99267399	-77.65567766
57249	1953	2103	2901	2046	1998	2055	1747.10083	-0.940594059	0.544554455	8.445544554	-56.77655678	-74.35897436	-77.65567766
57889	1960	2090	2902	2047	1999	2053	1766.63208	-0.871287129	0.415841584	8.455445545	-57.14285714	-74.72527473	-76.92307692
58529	1946	2092	2908	2044	1996	2053	1786.16333	-1.00990099	0.435643564	8.524752475	-56.04395604	-73.62637363	-76.92307692
59169	1958	2107	2911	2046	1999	2055	1805.69458	-0.891089109	0.584158416	8.544554455	-55.67765568	-74.35897436	-77.65567766

Figure 1. The neatly formatted file viewed in excel becomes a mess when viewed as plain text

The plain text file is in a series of comma delimited numbers in the same way that excel formats the file by column. Our initial thought was to simply read all of the data into memory, however, some quick math shows us why that is a bad idea.

Sampling Rate = 51 Hz
Line Buffer Size = 256 Bytes
$$Memory\ usage\ per\ second = \ 256\ Bytes * 51\ Hz = 13.056\ kB/s$$

In half an hour of use, storing every value from the file would use 23.5 MB of memory or almost 1 MB/min.

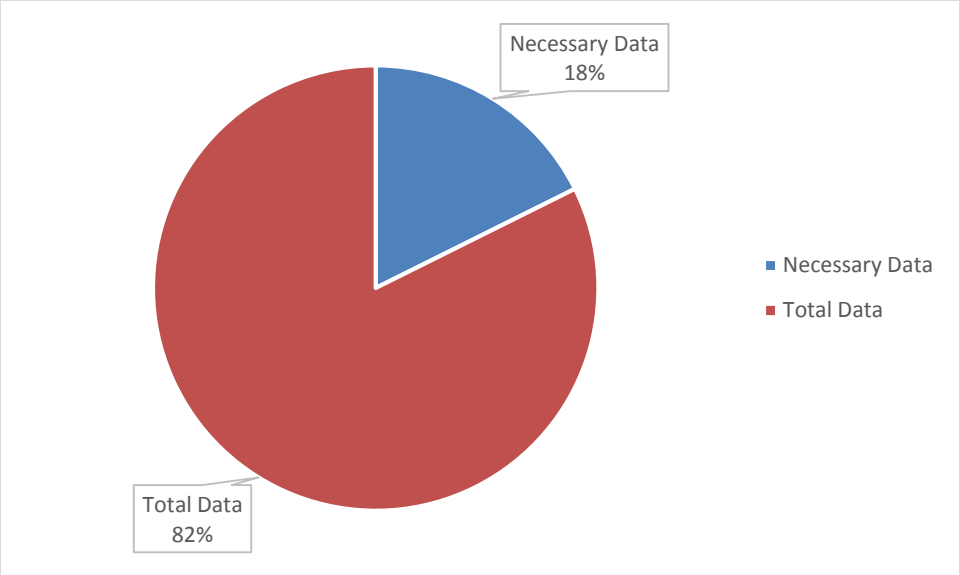


Figure 2. Necessary data only occupies 18% of the total data we're sampling

As we can see from figure 2, we only actually require 18% of the total amount of data written into the file. Therefore, by only storing values which we need, we can reduce the amount of memory we use by a factor of ~5.5!

$$Memory\ usage\ per\ second = \ 0.18 * 256\ Bytes * 51\ Hz = 2.35\ kB/s$$

Unfortunately this massive reduction in memory usage does not directly translate into increased performance due to the fact that memory is extremely large. However, this increased efficiency can be applied to more sophisticated systems where higher sampling rates provide more accurate readings and by extension, have higher memory overheads.

We had successfully read from the file and now we were faced with two additional challenges: translating the numbers into keystrokes and synchronizing the timing between reading the values and relaying the corresponding keystrokes to the program. Both of these issues are very important to the integrity of the project. The translation from head movement to keystroke needed to be a simple motion to pick up but robust enough to compete with a traditional video game controller. The synchronization of the timing between head movements and generated

keystrokes needed to be very precise or else the system would experience a large amount of continually increasing or decreasing latency.

In order to translate the numerical readings into keystrokes, we developed a two-factor threshold algorithm.

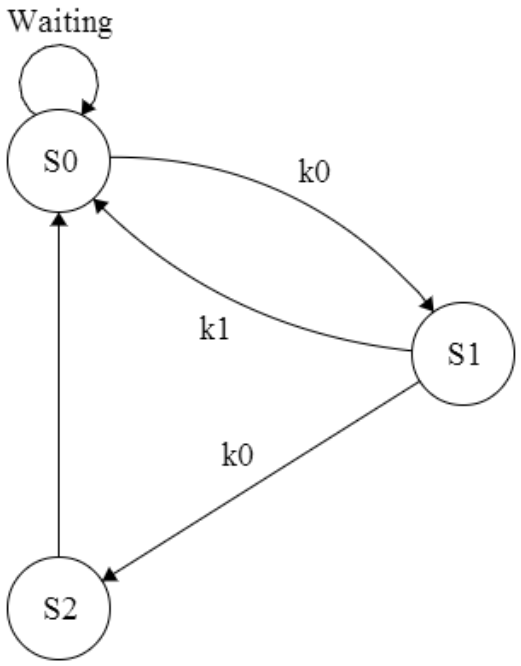


Figure 3. Finite state machine representation of our algorithm

In this algorithm, the system begins in the waiting state S0. If a key is generated, the system moves to state S1 and presses the key represented by K0. If in the next cycle a different key K1 is generated, the system releases K0 and begins to press K1. If the system is in state S1 and the same key K0 is generated again, then the system moves to state S2 where K0 is held for another cycle then the system returns to S0.

Figure 4. Pseudo-code representation of our algorithm

```
while(1){
      k = getKey(i);
      knext = getKey(i+1);
      if(k == knext){
            pressKey(k);
            // Wait for ~20ms
            releaseKey(k);
      }
      i = i + 2;
}
```

The most important part of our project was reducing the delay between the inputs and outputs as much as possible.

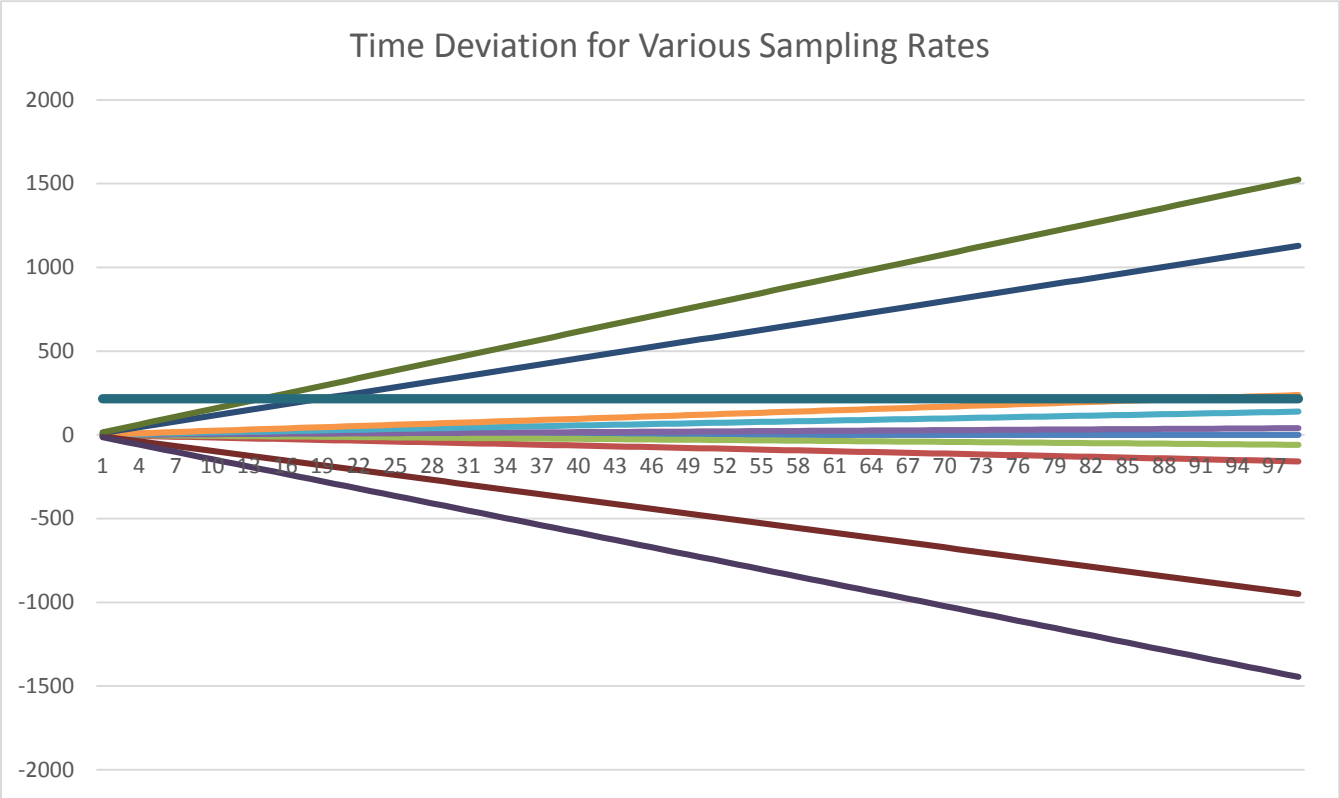


Figure 5. The extent at which varying sampling rates deviate from the input sampling rate inducing latency into the system. The horizontal black line indicates the median human reaction time at 215ms.

As we can see from the above figure, even a modest divergence from the exact sampling rate of 19.6ms can have profound effects over a period of time. The upwards sloping green line that appears to intersect the horizontal human reaction time line at t=77ms is the result of a constant 2.4ms delay between the device sampling rate and the software sampling rate. In other words, by using a sampling rate of 22ms instead of the exact 19.6ms, we introduce 237.6ms of delay after only 100ms.

Extrapolating from this data, we can look forward and find out exactly how much latency there will be in the system after a certain period of time with the following formula.

$$td = t * (r - rs)$$
$$td = time\ delay$$
$$t = time$$
$$r = software\ sampling\ rate$$
$$rs = device\ sampling\ rate\ (19.6ms)$$

For example, after 30 minutes of running with a 22ms sampling rate we would have a whopping 72 minutes of delay!

Cost and Sustainability Analysis

The cost of producing our posited solution to the issue of mobility is relatively low, due to the fact that most of our solution involves a software aspect. The only significant expense would be the shimmer sensors, which cost about $350 each. The use of the shimmer sensor is then all done on the software side, which doesn't bring up the cost of the product. In the future if we want to mass produce these it would much cheaper for us to use accelerometer and gyroscope computer chips as opposed to the prebuilt shimmer sensor. This would lower the cost of producing our product significantly, as you can get triple access accelerometer microchips for around $17 each but if bought in mass quantities it would cost pennies on the dollar. The same goes for the gyroscope microchips which are roughly $25 a piece but will be significantly cheaper if bought in large quantities. Thus allowing us to produce our capstone design fairly cheaply. As far as sustainability is concerned the product can continue to be produced as long as shimmer sensors (or at least a gyroscope and an accelerometer) are still available in mass quantities. Looking into future applications and possibly the addition of an EEG headset to allow for more control and accuracy, we can see that in the big picture the costs won't change by much. The EEG headset will cost around $750. The breakdown of total cost for products are as follows:

<u>Our demo implementation:</u>

$350 - per Shimmer Sensor
$750 - EEG headset
$200 - Laptop
$14000- Software Development

total = $15300

The Software development cost however only applies to the initial device development. The flat software development cost would not apply to each additional device after the first meaning it would cost $1300 for each additional device.

<u>Current technologies:</u>

Steven Hawking currently has arguably the most advanced wheelchair. Hawking's wheelchair works through a keyboard he can operate with his thumb. Hawking is a case where he still has the ability to operate a device with his hands. The technology in Hawking's wheelchair cost over $40,000

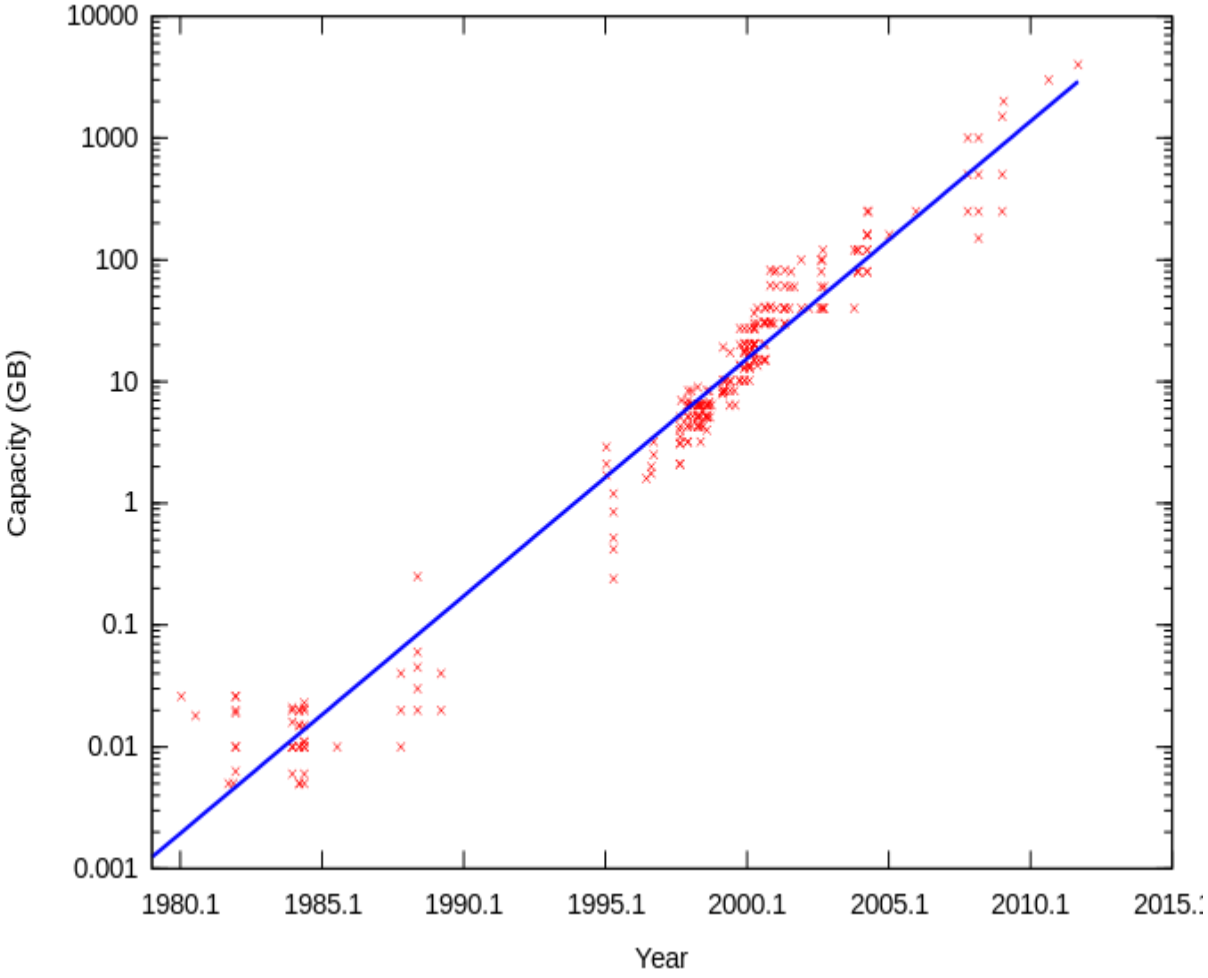
There is still room for improvement in our technology. Many modern phones contain gyroscopes, and accelerometers. These phones vary in price from $500 to $700. There are however many benefits of using a phone over a shimmer sensor. Getting the technology to reach the marketplace is significantly easier through the use of cell phones because 56% of Americans

already own smart phones. As phones get more advanced the cost to purchase a phone which supports the technology we require will become cheaper.

Our idea has very high sustainability. As time passes our idea becomes more feasible and there is room for improvement. As Ram capacity increases our ability to run the program and collect data improves.

*Memory Usage per second = 256 Bytes *51 bytes = 13.956kB/s*

In half an hour of use storing ever value to the file would use 23.5 mB. With current cost of Ram this ram usage is significant for running the program for long periods of time. With the increase in the availability of Ram however this data usage will become less and less significant.



You can see in the above figure that ram capacity has increased over the last 30 years

A problem that our implementation is currently facing is processing speed. The delay slowly increases over time due to the fact that the data is written to the file faster than it can be processed. This problem will become less of an issue as the number of cores increase on CPUs. Our test were performed on a intel quad core which is a relatively out of date CPU. The cores on the Intel i7 processor is 1.2 GHZ faster than on the quad core we used. Using an up to

date processor should easily remedy the small lag problem that currently exist. A problem with switching to a faster processor is a larger monetary cost.
The average price for an Intel i7 is $350 compared to the price of quad core we used which is approximately $100. This cost will be offset by the steady decrease in cost of processors. Implementing this technology seams very viable considering the cost are relatively low and are shown to decrease over time. It is our belief for this technology to be successful it should wait 5 years before it is released. As the power of smart phones improve and there availability increases. Gyroscope controlled technology will be more and more advantageous as the smart-phone market gross and its production cost increase.

Conclusion

All in all, it can be seen that we did in fact manage to fulfill the intended goals for the application of this project. By using a Shimmer Sensor we were able to read in gyroscope and accelerometer data in real time, which allowed us to determine which direction and to what extent the user was tilting his or her head. As described above, upon reading in the signals we parsed the data from the Shimmer Sensor such that we were only left with those pertinent to us. We then were able to determine the direction by using the data from the Shimmer Sensor to create control signals. These control signals each represented a distinct action in the game, and we were then able to apply the user's inputs to the Super Mario Bros. game by simulating these directions as keystrokes.

Though this is an interesting achievement in and of itself, what truly excited us was the prospect of being able to widen the scope of this project to more useful applications. For example controlling a wheelchair or perhaps turning on home appliances, for those who do not have the movement of their upper body. This project was heavily reliant upon Human Computer Interaction, and that is the direction that we would hope that this project could continue to go in the future. In this age more than ever, we are intertwined with the technology around us (i.e. smartphones, tablets etc.), and this unity with technology will only strengthen over time. It is for that reason that we want to continue down the path we are currently on.

Perhaps one day we can integrate the EEG device we mentioned earlier. This in turn would mean that the user could suffer from full body paralysis and still be able to function on a normal day to day level, controlling various electronics around them with just their thoughts. What we did here in our eyes was simply a first step of a much grander concept. This project acted as a proof of concept, which we could expand upon, continuing to use the Human Computer Interface model to create a piece of technology that seamlessly and cohesively works to ultimately help those who cannot move at all.

Bibliography

[1] "Shimmer Connect User Manual" Rev 0.11a.

[2] "Shimmer MATLAB Instrument Driver Library User Manual" Rev 1.8a.

[3] Burns, Adrian "SHIMMER™ – A Wireless Sensor Platform for Noninvasive Biomedical Research." *IEEE SENSORS JOURNAL*.

[4] Ingle, Vinay K., and John G. Proakis. *Digital Signal Processing Using MATLAB*. Pacific Grove, CA: Brooks/Cole, 2000. Print.

[5] Smith, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA: California Technical Pub., 1997. Print.

[6] *EEG*. Stuttgart: Thieme, 2010. Print.

[7] Sanei, Saeid, and Jonathon Chambers. *EEG Signal Processing*. Chichester, England: John Wiley & Sons, 2007. Print.

[8] EYE-EEG: Test Datasets." *EYE-EEG: Download Test Datasets*. EyeLink, n.d. Web. 05 May 2014.

Code:

(Controller.c, Contains the main function (program entry point), as well as functions that implement our two-factor threshold algorithm (converting numbers to keystrokes explained in my part of the report). Also contains functions that actually press the keys. This is the core of the program and also contains code that syncs the input to the output.)

```
#include "shimmer.h"
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

INPUT ip;
int count = 0;

// Presses key specified by virtual key code
int keydown(int key){
	ip.ki.wVk = key;
	ip.ki.dwFlags = 0;
	SendInput(1, &ip, sizeof(INPUT));
}

// Releases key specified by virtual key code
int keyup(int key){
	ip.ki.dwFlags = KEYEVENTF_KEYUP;
```

```
	SendInput(1, &ip, sizeof(INPUT));
}

// Translates a value to a virtual key code
int tk(char key){
	key = tolower(key);
	switch(key){
		case 0:
			return VK_UP;
		case 1:
			return VK_DOWN;
		case 2:
			return VK_LEFT;
		case 3:
			return VK_RIGHT;
		case 4:
			// Z - jump
			return 0x5A;
		case 5:
			// X - run/fireballs
			return 0x58;
		default:
			return 0;
	}
}


// function to evalute which key was pressed based on value
int sneakpeek(int j, int v){
	switch(j){
		case 0:
			if(v > 2100 && v < 2500){
				// walk left
				return 2;
			}
			else if(v > 1350 && v < 1800){
				// walk right
				return 3;
			}
			break;
		case 1:
			if(v <= 1800){
				// down/crouch
				return 1;
```

```
                }
                else if(v >= 2300){
                        // jump
                        return 4;
                }
                break;
        default:
                return -1;
                break;
    }
}
// function to evalute which key was pressed based on value
int eval(int j, int v){
    switch(j){
        case 0:
                if(v > 2100 && v < 2500){
                        // walk left
                        keydown(tk(2));
                        printf("walk left\n");
                        return 2;
                }
                else if(v > 1350 && v < 1800){
                        // walk right
                        keydown(tk(3));
                        printf("walk right\n");
                        return 3;
                }
                break;
        case 1:
                if(v <= 1800){
                        // down/crouch
                        keydown(tk(1));
                        printf("down\n");
                        return 1;
                }
                else if(v >= 2300){
                        // jump
                        keydown(tk(4));
                        printf("jump\n");
                        return 4;
                }
                break;
        default:
                return -1;
```

```
                break;
        }
}

int vtok(int n){
        int i, j, x, xnext, y, ynext;
        int cur[2];
        int next[2];

        for(i = 0; i < n - 1; i++){
                x = valuesx[i];
                xnext = valuesx[i + 1];
                y = valuesy[i];
                ynext = valuesy[i];

                cur[0] = eval(0, x);
                next[0] = sneakpeek(0, xnext);

                cur[1] = eval(1, y);
                next[1] = sneakpeek(1, ynext);

                Sleep(20);
                for(j = 0; j < 2; j++){
                        if(cur[j] != next[j]){
                                keyup(tk(cur[j]));
                        }
                }
        }
}

int main(){
        // Wait 1 second, gives us time to get into our program
        Sleep(1000);
        offset = 0;

        ip.type = INPUT_KEYBOARD;
        ip.ki.wScan = 0;
        ip.ki.time = 0;
        ip.ki.dwExtraInfo = 0;

        // Get rid of junk data

        valuesx = malloc(sizeof(int) * 3000);
        valuesy = malloc(sizeof(int) * 3000);
```

```
	int numvals = 0;
	initRead();
	while(1){
		numvals = shimmerRead();
		vtok(numvals);
	}
	return 0;
}
```

(Shimmer.c, Contains functions that read the file and parse the data. Basically, we open the file, seek to the end, then save the position of the file pointer. This synchronizes our program with the shimmerconnect program so the next data we read is the most recent data written to the file.

Then we pass the string we read to the parse function so that we're only storing the data that we need.)

```
#include "shimmer.h"

#define LINESIZE 256

int shimmerRead(){
	char *buffer = malloc(sizeof(char) * LINESIZE);
	int i;
	int offset2;
	long sum = 0;
	int numlines = 0;

	do{
	fp = fopen("mario.csv","r");
		fseek(fp,0,SEEK_END);
		offset2 = ftell(fp);
		if(offset2<(offset+1000)){
			fclose(fp);
		}
	}while(offset2<(offset+1000));

	fseek(fp,offset,SEEK_SET);

	// Get rid of garbage header
	for(i = 0; i < 4; i++){
		fgets(buffer, LINESIZE, fp);
	}
```

```
	while(fgets(buffer, LINESIZE, fp) != NULL){
		shimmerParse(buffer, numlines);
		numlines++;
	}

	free(buffer);

	offset = ftell(fp);
	numlines--;
	fclose(fp);
	return numlines;
}

void shimmerParse(char *s, int n){
	int counter = 0;
	char *pch = malloc(sizeof(char) * 50);

	pch = strtok(s, ",");
	do{
		if(counter == 1){
			//printf("%s\n", pch);
			valuesx[n] = atoi(pch);
		}
		else if(counter == 2){
			valuesy[n] = atoi(pch);
		}
		pch = strtok(NULL, ",");
		counter++;
	}while(counter < 3);
}


void initRead(){
	// Open file, seek to end, save fp position in 'offset'

fp = fopen("mario.csv","r");

fseek(fp,0,SEEK_END);
offset = ftell(fp);

fclose(fp);
}
```

(Shimmer.h, Contains the function prototypes for shimmer.c as well as variable declarations for variables shared between shimmer.c and controller.c)

```
#ifndef SHIMMER_H_
#define SHIMMER_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

FILE *fp;
long offset;

int *valuesx;
int *valuesy;

int **values;

void initRead();

int shimmerRead();
void shimmerParse(char *s, int n);

#endif
```

(Makefile, tool to easily compile our code using commands like 'make all' and 'make control' or 'make shimmer')

```
# alias that exists so 'make all' compiles the control program
all: control

# compile/link to one executable
control: controller.o shimmer.o
	gcc controller.o shimmer.o -o control

# compile only the controller.c file (for performance purposes)
controller.o: controller.c shimmer.h
	gcc -c controller.c

# compile only the shimmer.c file (for performance purposes)
shimmer.o: shimmer.c shimmer.h
	gcc -c shimmer.c

# remove compiled files
```

```
clean:
	rm -rf *.o
	rm -f control.exe
```